OPL - PathImpact

Analyseur automatique d'impacts de Code Source

UNIVERSITÉ LILLE 1 – SCIENCES ET TECHNOLOGIES

13 février 2017

Créé par : Étienne WATTEBLED

OPL - PathImpact

Analyseur automatique d'impacts de Code Source

Table des matières

I. Introduction

De jour en jour, afin d'améliorer la productivité ou l'efficacité, les métiers évoluent provoquant des besoins fonctionnels qui ne cessent de changer même lorsque les applications sont opérationnelles et utilisées par les utilisateurs. Les applications possèdent, par conséquent, de plus en plus de fonctionnalités et deviennent de plus en plus complexes. Très vite, la maintenance devient alors très onéreuse, difficile à gérer et constitue l'une des plus grandes préoccupations des développeurs. Il n'est pas rare qu'une ou plusieurs fonctionnalités soient modifiées, ou même supprimées durant le développement d'un logiciel, ce qui peut provoquer de multiples modifications qui se répercutent.

On appelle ces modifications des « impacts ».

Ces derniers peuvent devenir rapidement catastrophiques, de par l'apparition de régressions ou de code simplifiable voire mort qui peut persister pendant des années. De plus, à cause de la complexité et de la taille de certaines applications, certaines modifications peuvent s'avérer extrêmement coûteuses à mettre en place, si bien que, le client pourrait refuser de procéder à une évolution.

Il est alors indispensable de pouvoir évaluer avec précision, les impacts provoqués par une modification donnée d'une méthode, ne serait-ce que pour estimer le coût ou conserver le code sans parasites. Cependant, l'analyse manuelle peut s'avérer complexe et longue, et si l'évolution est annulée, ce temps utilisé aura été inutilement perdu.

Une solution existe : en effet il existe des algorithmes capables de détecter automatiquement les impacts, plus précisément, de lister toutes les méthodes susceptibles d'être « touchées » par la modification d'une méthode. Ils sont utilisés pour faire des chiffrages, pour gagner du temps à mettre à jour une STD (Spécifications Techniques Détaillées), ou encore, lister avec précision les modifications à apporter avant de procéder au développement pour ne rien oublier.

Certains analyseurs d'impacts sont plus précis que d'autres mais les algorithmes permettant de parvenir à des résultats extrêmement précis sont très consommateurs en termes de puissance de calculs.

Ce rapport présente une grande partie de l'implémentation d'un analyseur d'impacts appelé « PathImpact » qui fait partie des analyseurs qui consomment le plus de ressources, mais il a l'avantage de fournir une excellente précision. Seule l'étape permettant de déterminer les impacts n'a pas été traitée dans ce projet.

Avant tout, ce document explique comment fonctionne PathImpact et détaille par la suite son implémentation. La troisième partie concerne les tests unitaires JUnit, suivie d'un chapitre traitant l'évaluation. Et enfin, une phase de critiques et de limitations précède la conclusion de ce travail.

II. PathImpact

PathImpact est un analyseur d'impacts fonctionnant en trois étapes principales :

1- **Détermination et concaténation des stacktraces**

2- **Compression des stacktraces via un algorithme appelé Sequitur**

3- **Construction d'un arbre appelé DAG, et détermination des impacts.**

Ces étapes sont expliquées dans les parties ci-dessous.

a. Stacktraces

L'algorithme PathImpact a besoin des différentes stacktraces en entrée pour déterminer les impacts.

Afin de les déterminer, il est nécessaire d'exécuter le programme dans tous ses états possibles, autrement dit, tester tous les cas de paramètres en entrée. Ainsi, si le programme a juste besoin d'un entier « n » pour fonctionner, il faut tester toutes les valeurs de « n » représentables (de Integer.MIN_VALUE jusque Integer.MAX_VALUE en Java).

La stacktrace est représentée par une collection ordonnée d'éléments représentant les méthodes. Ainsi, « A B » signifie « la méthode A appelle la méthode B ». Cependant, pour pouvoir faire la différence entre : « A appelle B, B retourne une valeur, puis A appelle C » et « A appelle B, puis B appelle C », il est nécessaire d'intégrer un nouvel élément que l'on appelle « r » et qui signifie « return ». Ainsi, « ABrCDrrr » signifie : « A appelle B, B retourne une valeur, A appelle C, C appelle D, D retourne une valeur, C retourne une valeur, puis A retourne une valeur ».

Toutes les stacktraces sont par la suite concaténées les unes après les autres. Par conséquent, un nouvel élément « x » est utilisé et signifie « fin du programme ».

Le fait que la stacktrace soit une liste d'éléments, permet plus de possibilités par la suite.

En considérant l'exemple ci-dessous, la stacktrace générée doit alors être :

« main, m, r, r, x, main, m2, m3, r, m4, r, r, r, x »

```
public class Test {
        public static void main(String args[]) {
                int i = 0;
                i = Integer.parseInt(args[i]) ;
                if (i == 0) {
                        m();
                } else {
                        m2();
                }
        }
        public static void m() {
                …
        }
        public static void m2() {
                m3();
                m4();
        }
        public static void m3() {
                …
        }
        public static void m4() {
                …
        }
}
```

b. Sequitur

Sequitur est un algorithme de compression utilisant la théorie des langages, plus précisément, la grammaire pour parvenir à ses fins.

Une grammaire est un ensemble de règles de production qui permet de générer les mots d'un langage. Toute grammaire possède une règle de départ (axiome) que l'on appelle la plupart du temps « S ».

Voici par exemple, la grammaire des expressions arithmétiques, dans laquelle l'axiome est « exp » :

exp ⟶ *exp* + *exp* | *exp* × *exp* | (*exp*) | *num*
num ⟶ *chiffre num* | *chiffre*
chiffre ⟶ *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*

Wikipédia (Grammaire formelle)

L'algorithme de Sequitur est basé sur deux propriétés fondamentales :

1) Il ne doit pas y avoir de doublons de digramme.
   Définition de digramme : assemblage de deux signes (généralement deux lettres de l'alphabet).

2) Si une règle de production n'est plus utilisée ou n'est utilisée qu'une seule fois, elle doit être « désappliquée » puis supprimée.

Ainsi, la grammaire G définit par les règles :

S → aaaabb1p
1 → cde
2 → fgh

déroge les deux propriétés (1 et 2) car :

1) Le digramme « aa » est présent deux fois.
   Solution : créer une nouvelle règle 3 → aa

2) La règle 1 n'est utilisée qu'une seule fois.
   La règle 2 n'est pas du tout utilisée.
   Solution : supprimer la règle 2 et « désappliquer » la règle 1.

Une des grammaires équivalentes respectant les deux propriétés serait :

S → 33bbcdep
3 → aa

Étant donné que la grammaire vide respecte les deux propriétés, celle-ci devient le point de départ de Sequitur et la règle de production « S » va grandir petit à petit.

Dès qu'une règle est dérogée, Sequitur va utiliser les solutions associées aux propriétés (ci-dessus) pour rectifier la grammaire.

Le pseudo algorithme de Sequitur est donc :

En paramètre, une simple liste l d'éléments

. Créer une grammaire g vide.
. Ajouter le 1er élément de la liste l à la règle S de la grammaire g.
. Pour chaque élément e de la liste l (à partir du 2ème élément)
    . Ajouter l'élément e dans la règle S de la grammaire (à la fin)
    . Tant que les deux derniers éléments de S forment un digramme déjà rencontré
        . S'il existe déjà une règle r qui produit ces deux éléments
            . La récupérer
        Sinon
            . Créer la règle r qui produit ces deux éléments
        Fin si
        . Appliquer la règle r sur toute la grammaire
        . Désappliquer puis supprimer toutes les règles qui ne sont pas utilisées au moins deux fois
    Fin tant que
Fin pour

Ce pseudo algorithme est simplifié, principalement sur la notion de « digramme déjà rencontré ».

Exemple : On considère les éléments suivants :

« D A B A B C D A B E B C D A B A B »

S →

S → D

S → D A

S → D A B

S → D A B A

S → D A B A B [le digramme A B existe, on crée une règle]

S → D 1 1
1 → A B

S → D 1 1 C
1 → A B

S → D 1 1 C D
1 → A B

S → D 1 1 C D A
1 → A B

S → D 1 1 C D A B [la règle AB existe déjà, on l'utilise]
1 → A B

S → D 1 1 C D 1 [le digramme D1 est en double, on crée une règle]
1 → A B

Tant que les deux derniers éléments sont en double, aucun élément ne doit être ajouté à « S ».

S → 2 1 C 2
1 → A B
2 → D 1

S → 2 1 C 2 E
1 → A B
2 → D 1

S → 2 1 C 2 E B
1 → A B
2 → D 1

S → 2 1 C 2 E B C
1 → A B
2 → D 1

S → 2 1 C 2 E B C D
1 → A B
2 → D 1

S → 2 1 C 2 E B C D A
1 → A B
2 → D 1

S → 2 1 C 2 E B C D A B [la règle AB existe]
1 → A B
2 → D 1

S → 2 1 C 2 E B C D 1 [la règle D1 existe]
1 → A B
2 → D 1

S → 2 1 C 2 E B C 2 [la règle C 2 doit être créée]
1 → A B
2 → D 1

S → 2 1 3 E B 3
1 → A B
2 → D 1
3 → C 2

S → 2 1 3 E B 3 A
1 → A B
2 → D 1
3 → C 2

S → 2 1 3 E B 3 A B [règle A B existe]
1 → A B
2 → D 1
3 → C 2

S → 2 1 3 E B 3 1
1 → A B
2 → D 1
3 → C 2

Sequitur fonctionne donc sur des digrammes et parvient parfois à créer des règles de plus de deux éléments. Mais comment parvient-il à réutiliser une règle de taille supérieure à 2 ?

En réalité, Sequitur va former une nouvelle règle qui va progressivement grandir pendant que la règle qui existait va progressivement diminuer de taille jusqu'à n'avoir plus que deux éléments. À partir de là, puisque cette règle n'aura plus que deux éléments, elle sera détectée et la dernière règle créée ne sera plus utilisée qu'une seule fois, elle sera donc supprimée.

Pour illustrer ce phénomène, voici un exemple :

« ABCDABCDABCD »

S → A

S → A B

S → A B C

S → A B C D

S → A B C D A

S →A B C D A B [création règle AB]

S → 1 C D 1
1 → A B

S → 1 C D 1
1 → A B

S → 1 C D 1 C [création règle 1C]
1 → A B

S → 2 D 2
1 → A B [règle 1 utilisée une seule fois]
2 → 1 C

S → 2 D 2
2 → A B C

S → 2 D 2 D [création règle 2D]
2 → A B C

S → 3 3
2 → A B C [règle 2 utilisée une seule fois]
3 → 2 D

S → 3 3
3 → A B C D

S → 3 3 A
3 → A B C D

S → 3 3 A B [AB est en doublon même s'il est dans une règle : on crée une nouvelle règle]
3 → A B C D

S → 3 3 4
3 → 4 C D
4 → A B

La règle 3 va progressivement diminuer de taille, pendant que la règle 4 va grandir.

S → 3 3 4 C [4C est en doublon, création d'une nouvelle règle]
3 → 4 C D
4 → A B

S → 3 3 5
3 → 5 D
4 → A B [la règle 4, utilisée une seule fois]
5 → 4 C

S → 3 3 5
3 → 5 D
5 → A B C

S → 3 3 5 D [la règle 5 D existe]
3 → 5 D
5 → A B C

S → 3 3 3
3 → 5 D
5 → A B C [la règle 5 n'est utilisée qu'une seule fois

S → 3 3 3
3 → A B C D

On constate que Sequitur a bien réussi à réutiliser une règle possédant plus de deux éléments, et ce, grâce à la fois à la création de nouvelles règles de deux éléments et aux simplifications.

Afin d'éviter les règles en multiples exemplaires, il est donc important de vérifier si le digramme à la fin de S n'est pas non plus présent dans une des règles (en doublon).

De plus, il est impossible d'obtenir des doublons de digramme, car, que ce soit dans « S » ou dans une règle, si le digramme existe déjà alors une règle est obligatoirement créée.

Sequitur est donc un algorithme qui fonctionne correctement dans tous les cas et la compression est très efficace.

c. DAG (Directed Acyclic Graph)

Le DAG est une représentation de la grammaire de Sequitur sous forme d'arbre de profondeur 2.

La racine contient tous les éléments de « S », la couche du dessous contient toutes les règles de production, et les feuilles sont les autres éléments (x, r, et les méthodes). Chaque élément de la racine est, soit rattaché à la couche du dessous s'il s'agit d'une règle, soit rattaché à une feuille. Tous les éléments des règles sont aussi rattachés de la même manière.

Une fois de DAG créé, si une méthode est modifiée, il est possible de déterminer les impacts. Pour cela, l'algorithme commence sur la feuille de la méthode modifiée et parcourt le DAG en avançant tant que l'élément « x » n'est pas présent. En procédant de cette manière, PathImpact détermine toutes les méthodes appelées après la méthode modifiée, et donc, susceptibles d'être affectées par le changement.

III. Implémentation

a. Structures principales

Stacktrace

La stacktrace est représentée sous la forme d'une liste chaînée d'éléments.

Les classes et interfaces utilisées pour représenter une stacktrace sont les suivantes :

- **ElementItf** : interface représentant tous les éléments d'une stacktrace.
  Seules les méthodes getNom() et print() sont présentes dans cette interface.

- **AbstractElement** : Classe abstraite implémentant ElementItf et ne possédant uniquement que le nom.

- **Evenement** : Type énuméré implémentant ElementItf. Les seules valeurs possibles sont « RETURN » dont le nom est « r » ainsi que « END_OF_PROGRAM » dont le nom est « x ».

- **Methode** : Classe héritant de AbstractElement.

Grammaire

Afin d'implémenter Sequitur, une classe Grammaire a été mise en place.

Il est ainsi possible :

- De créer une grammaire

- D'ajouter un élément à « S »

- D'appliquer une règle à « S » ou à la grammaire entière.

- De désappliquer une règle à « S » ou à la grammaire entière.

- De récupérer toutes les règles de la grammaire

- De simplifier toutes les règles.

- De connaître l'existence d'une règle placée en paramètre

- D'afficher la grammaire (via une méthode print, affichage axiome + règles)
- De créer une règle
- De supprimer une règle (la règle sera alors automatiquement désappliquée sur toute la grammaire)
- De récupérer tous les éléments de « S » ou d'une règle.

Une classe Règle, permettant de représenter une règle a été créée et celle-ci hérite de la classe AbstractElement. Il a été choisi lors de l'implémentation que celle-ci soit une classe interne à Grammaire afin qu'aucune règle ne puisse, entre autres, être créée en dehors d'une grammaire. De plus, le fait que la classe Règle soit interne permet de ne rendre visibles certaines méthodes que pour la grammaire (exemple : le constructeur). De plus, il est plus juste de considérer le fait qu'il ne peut pas exister de règle sans grammaire. L'élément « S » (axiome) est en fait, représenté en tant que règle. Une fois l'instance de la règle obtenue par la grammaire, il est possible d'accéder aux éléments de celle-ci. Cependant, afin que les utilisateurs ne puissent pas modifier les éléments d'une règle sans passer par la grammaire, ou encore, supprimer une règle de la grammaire en récupérant la liste de toutes les règles sans que la grammaire en soit informée (et donc bypasser le système), toutes les listes sont retournées via une vue non modifiable (il existe des méthodes dans la classe Collections en Java qui permettent de retourner une vue non modifiable d'une liste, d'une map… etc.)

b. Détermination des Stacktraces

Afin de récupérer les stacktraces, il est extrêmement complexe voire impossible de tester dans un temps raisonnable toutes les combinaisons de données possibles en paramètres et de déterminer toutes les stacktraces possibles du programme.

Une solution consiste à faire un parcours en profondeur et, une fois arrivé sur une feuille, de considérer le programme comme étant terminé. Il ne faut donc pas oublier de générer les éléments « r » correspondant ainsi l'élément « x » à la fin de chaque stacktrace.

Ainsi, en reconsidérant la classe Test vue précédemment (page 7), la stacktrace sera :

« main m r r x main m2 m3 r r r x main m2 m4 r r r x »

au lieu de

« main, m, r, r, x, main, m2, m3, r, m4, r, r, r, x »

Pour générer correctement ces stacktraces et les concaténer, Spoon a été utilisé via un processor qui se déclenche sur chaque méthode (CtMethod). Si la fonction est une fonction main, alors elle est ajoutée à une liste dédiée aux fonctions mains. Par la suite, toutes les invocations présentes dans la méthode (tous

les appels de méthode) sont enregistrées dans une Map qui, pour une méthode donnée, fournit une liste de méthodes appelées.

Lorsque le processor a terminé, il ne doit y avoir qu'une seule et unique fonction main dans la liste des fonctions mains (sinon, une erreur survient). Il suffit ensuite de commencer de la fonction main et de faire un parcours en profondeur récursif.

Attention, trois cas particuliers sont à prévoir :

- Il faut ignorer les méthodes abstraites car celles-ci n'appellent aucune méthode (il n'y a pas de code).

- Il faut gérer les appels récursifs dans le parcours en profondeur. Si l'élément, lors du parcours en profondeur, est déjà dans la stacktrace en cours de création, alors il doit être ajouté dans la stacktrace et cette dernière doit être formée (et donc, il faut considérer que cela forme une exécution complète). Il faut ensuite continuer avec les prochains éléments…

- Si une méthode appelle une méthode abstraite (méthode d'interface ou de classe abstraite), il faut rechercher dans le projet toutes les implémentations correspondantes.

En considérant la classe ci-dessous, la stacktrace est :

« main, a, a, r, r, r, x, main, a, c, r, r, r, x, main, b, b, r, r, r, x »

```
public class Test {
        public static void main(String args[]) {
                a();
                b();
        }
        public static void a() {
                a();
                c();
        }
        public static void b() {
                b();
        }
        public static void c() {
                …
        }
}
```

c. Sequitur

Une fois la grammaire implémentée, il a été possible d'implémenter plus facilement Sequitur.

Lors de l'application d'une règle, un entier correspondant au nombre de fois où la règle a été appliquée est remonté.

- Si ce nombre est inférieur à 2, la simplification des règles va supprimer (et donc désappliquer) automatiquement cette règle.

- Si ce nombre est égal à 2, alors une nouvelle tentative d'application de la règle suivante (2 derniers éléments de « S ») peut avoir lieu.

Attention, si les deux derniers éléments forment une règle existante, alors le nombre de fois où la règle a été appliquée est égal à 1 : il est alors nécessaire d'essayer de compresser davantage et ne surtout pas désappliquer la règle, ni ajouter un élément supplémentaire dans « S ».

IV. Tests unitaires JUnit

Afin d'éviter toute régression future, les tests unitaires JUnit sont fondamentaux.
Des tests unitaires JUnit ont été implémentés sur les trois parties principales du projet, c'est-à-dire :

- Le processor Spoon permettant de générer les stacktraces
- La classe Grammaire
- La classe Sequitur

Les tests unitaires implémentés couvrent la quasi-totalité de PathImpact (seules les méthodes trop triviales telles que l'ajout d'un élément dans « S », n'ont pas été testées). Toutes les méthodes sensibles ont été testées.

Les tests unitaires jouent aussi le rôle de pré-évaluation.

a. Tests sur les stacktraces

La classe de tests permettant de tester la génération des stacktraces possède cinq tests.

Lorsqu'aucune, ou plus d'une fonction main est détectée, PathImpact lance un System.exit(0) et génère un message d'erreur dans le canal des erreurs. Il était important de pouvoir tester l'appel à la méthode exit(0), ainsi que l'apparition du message dans le canal d'erreurs.

Pour cela, une librairie complémentaire à JUnit a été mise en place sur le projet permettant de faire des tests via des règles systèmes (la librairie s'appelle System Rules).

Les tests de la génération des stacktraces s'exécutent sur des projets qui ne contiennent, en majeure partie, qu'une ou deux classes dont les méthodes sont vides (hormis les appels).

Le premier test s'exécute sur un projet ne disposant pas de fonction main, le second s'exécute sur un projet disposant de deux fonctions mains. Ils permettent de vérifier que PathImpact appelle la méthode System.exit(0) et génère bien un message d'erreur dans le canal d'erreurs.
Le troisième s'exécute sur un projet fonctionnel et vérifie la stacktrace générée, l'avant dernier permet de tester la stacktrace avec des appels de méthodes abstraites, et le dernier vérifie le comportement de la stacktrace avec de la récursivité.

b. Tests sur la grammaire

Des tests unitaires ont aussi été mis en place sur la classe de Grammaire.
Ils consistent principalement à tester :

- L'ajout d'une règle
- La suppression d'une règle
- L'application d'une règle sur « S » et le nombre de fois où elle est appliquée
- La désapplication d'une règle sur « S » et le nombre de fois où elle est désappliquée.
- La simplification des règles

c. Tests sur Sequitur

Afin d'exécuter des tests unitaires sur l'algorithme de Sequitur, les exemples présents sur les sources [1], [2] et [3] ont été implémentés.

Seul le test unitaire sur la source [1] est différent. La seule différence est que, par rapport au document, la règle qui produit « C E r » existe mais celle-ci n'est pas dans le document. Cependant, d'après l'implémentation en ligne source [4], cette règle apparaît bien. De plus, « C E r » est un trigramme présent 2 fois, cela déroge donc une des deux propriétés de Sequitur.

V. Évaluation

La phase d'évaluation s'est effectuée en deux temps :

- Avant d'exécuter la génération des stacktraces ainsi que l'exécution de Sequitur sur un gros projet, PathImpact a d'abord été lancé sur quelques exemples dont certains ont été, par la suite, implémentés dans les tests unitaires.

- Une fois la première partie validée, une évaluation sur le projet JSoup a été effectuée.

a. Via des exemples

Génération des stacktraces

1er exemple :

```
package pathimpact ;

public class Test {
	public static void main(String args[]) {
		m();
		m2();
		m5();
	}
	public static void m() {
		m3();
	}
	public static void m2() {
		m4();
	}
	public static void m3() {

	}
	public static void m4() {

	}
	public static void m5() {
	}
}
```

Résultat : pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m() pathimpact.Test#m3() r r r x pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m2() pathimpact.Test#m4() r r r x pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m5() r r x

2ème exemple :

```
package pathimpact ;

public class Test {
	public static void main(String args[]) {
		m();
		m5();
	}
	public static void m() {
		m2();
	}
	public static void m2() {
		m3();
	}
	public static void m3() {
		m();
		m4() ;
	}
	public static void m4() {

	}
	public static void m5() {
		m4();
		m5();
	}
}
```

Résultat : pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m() pathimpact.Test#m2() pathimpact.Test#m3() pathimpact.Test#m() r r r r r x pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m() pathimpact.Test#m2() pathimpact.Test#m3() pathimpact.Test#m4() r r r r r x pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m5() pathimpact.Test#m4() r r r x pathimpact.Test#main(java.lang.String[]) pathimpact.Test#m5() pathimpact.Test#m5() r r r x

Exemple source [1] :

« MBGrGrrACErrrxMBCrGrrxMACErDrrrxMBCErFrDrrCErFrDrrrx »

Résultat attendu :

S → 2 1 3 6 7 x 2 C r 3 8 6 9 r 8 B 14 14 r x
1 → G r
2 → M B
3 → 1 r
6 → A C E r
7 → r r
8 → x M
9 → D 7
14 → C E r F r 9

Résultat obtenu :

```
S → 15 4 21 30 34 x 15 C r 21 38 30 50 r 38 B 62 62 r x
4 → G r
15 → M B
21 → 4 r
30 → A 44
34 → r r
38 → x M
44 → C E r
50 → D 34
62 → 44 F r 50
```

La grammaire obtenue est équivalente, seule la règle qui produit « C E r » est supplémentaire mais le résultat est cohérent car le trigramme est en double. De plus, en testant cet exemple sur la source [4], le résultat obtenu est :

0 → 1 2 3 4 5 x 1 C r 3 6 4 7 r 6 B 8 8 r x
1 → M B
2 → G r
3 → 2 r
4 → A 9

5 → r r
6 → x M
7 → D 5
8 → 9 F r 7
9 → C E r

Exemple source [2] :

« bbebeebebebbebee »

Résultat attendu :

S → D B D
A → b e
B → A A
D → b B e

Résultat obtenu :

S → 17 9 17
3 → b e
9 → 3 3
17 → b 9 e

La grammaire est équivalente.

Exemple source [3]

« M B r A C D r E r r r r x M B G r r r x M B C F r r r r x »

Résultat attendu :

S → 2 r A C D r E 1 1 3 G 4 3 C F 4 r x
1 → r r
2 → M B
3 → x 2
4 → 1 r

Résultat obtenu :

S → 14 r A C D r E 10 10 22 G 28 22 C F 28 r x
10 → r r
14 → M B
22 → x 14
28 → 10 r

La grammaire est équivalente.

b. Sur JSoup

Sur le projet JSoup, deux fonctions mains sont présentes. Afin de pouvoir tester PathImpact sur Jsoup, il faut par conséquent commenter une des deux fonctions mains :

- org.jsoup.examples.HtmlToPlainText
- org.jsoup.examples.ListLinks

Fonction main de la classe HtmlToPlainText

La stacktrace de départ contenait 1086 éléments. Après l'exécution de Sequitur, elle ne contenait plus que 285 éléments. La grammaire obtenue est la suivante :

S => org.jsoup.examples.HtmlToPlainText#main(java.lang.String[])
org.jsoup.helper.Validate#isTrue(boolean, java.lang.String) 27
org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 47
org.jsoup.Connection$Method#hasBody() 47 org.jsoup.helper.Validate#isFalse(boolean, java.lang.String) 47
org.jsoup.helper.HttpConnection$Response#serialiseRequestUrl(org.jsoup.Connection$Request)
org.jsoup.helper.Validate#isFalse(boolean, java.lang.String) 73 93 109
org.jsoup.helper.HttpConnection#needsMultipart(org.jsoup.Connection$Request) 93 115 109
org.jsoup.helper.DataUtil#mimeBoundary() 151
org.jsoup.helper.HttpConnection$Response#initUnSecureTSL() 151
org.jsoup.helper.HttpConnection$Response#getInsecureVerifier() 151
org.jsoup.Connection$Method#hasBody() 151
org.jsoup.helper.HttpConnection$Response#getRequestCookieString(org.jsoup.Connection$Request) 237 237 257 org.jsoup.helper.DataUtil#crossStreams(java.io.InputStream, java.io.OutputStream) 298
org.jsoup.helper.HttpConnection$Response#createHeaderMap(java.net.HttpURLConnection) 298

327 357 353 327 org.jsoup.parser.TokenQueue#matchChomp(java.lang.String)
org.jsoup.parser.TokenQueue#matches(java.lang.String) 511 357 406
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 406 484 626 540 626 353
org.jsoup.helper.HttpConnection$Base#hasCookie(java.lang.String) 697 697 761
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 213
org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 761
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 848
org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 213 826 823 826 848 1122 782
org.jsoup.helper.StringUtil#resolve(java.net.URL, java.lang.String) 903
org.jsoup.helper.HttpConnection#encodeUrl(java.net.URL) 903
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request,
org.jsoup.helper.HttpConnection$Response) 903
org.jsoup.helper.HttpConnection$Response#contentType() 903 org.jsoup.parser.Parser#xmlParser()
903 org.jsoup.helper.DataUtil#getCharsetFromContentType(java.lang.String)
org.jsoup.helper.DataUtil#validateCharset(java.lang.String) 213 998 985 1024 1081 1024 1119 823
1069 org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 1081 1094
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 1119 1122 1035 87
org.jsoup.helper.DataUtil#readToByteBuffer(java.io.InputStream, int)
org.jsoup.helper.Validate#isTrue(boolean, java.lang.String) 213
org.jsoup.helper.DataUtil#emptyByteBuffer() 115 1176
org.jsoup.helper.HttpConnection#timeout(int) 1176
org.jsoup.helper.HttpConnection#userAgent(java.lang.String)
org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 93 1200
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 1208 1200
org.jsoup.helper.HttpConnection#encodeUrl(java.lang.String)
org.jsoup.helper.HttpConnection#encodeUrl(java.net.URL) 1224 1237
org.jsoup.nodes.Node#childNodeSize() 1258 org.jsoup.nodes.Node#childNode(int) 1300 1258
org.jsoup.nodes.Node#parentNode() 1300 1308 1374 1334 org.jsoup.nodes.Node#childNode(int)
1356 1334 org.jsoup.nodes.Node#parentNode() 1356 1426 1374 1393
org.jsoup.nodes.Node#childNode(int) 1413 1393 org.jsoup.nodes.Node#parentNode() 1413 1426 x
R = {
11 => r r
27 => 11 87
47 => 73 27
73 => 11 11
87 => 1170 org.jsoup.helper.HttpConnection#get() org.jsoup.helper.HttpConnection#execute()
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request)
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request,
org.jsoup.helper.HttpConnection$Response)
93 => 11 r
109 => 87
org.jsoup.helper.HttpConnection$Response#setOutputContentType(org.jsoup.Connection$Request)
115 => 93 r

151 => 213
org.jsoup.helper.HttpConnection$Response#createConnection(org.jsoup.Connection$Request)
213 => 310 87
237 => 257 org.jsoup.helper.HttpConnection#encodeMimeName(java.lang.String)
257 => 213 org.jsoup.helper.HttpConnection$Response#writePost(org.jsoup.Connection$Request,
java.io.OutputStream, java.lang.String)
298 => 213
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
310 => 115 93
327 => org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
org.jsoup.parser.TokenQueue#chompTo(java.lang.String)
353 => 310 93 671
357 => org.jsoup.parser.TokenQueue#consumeTo(java.lang.String)
org.jsoup.parser.TokenQueue#remainder()
406 => 431 org.jsoup.helper.HttpConnection$Base#cookie(java.lang.String, java.lang.String)
431 => 310 27
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
458 => 431 org.jsoup.helper.HttpConnection$Base#header(java.lang.String, java.lang.String)
484 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 458
511 => 353 org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
540 => 511 org.jsoup.helper.HttpConnection$Base#header(java.lang.String, java.lang.String)
626 => org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 458 484
org.jsoup.helper.HttpConnection$Base#removeHeader(java.lang.String)
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 540
org.jsoup.helper.HttpConnection$Base#removeHeader(java.lang.String)
org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String)
671 => 87
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
697 => 985 671 org.jsoup.helper.HttpConnection$Base#cookie(java.lang.String, java.lang.String)
707 => 310 r
761 => 823 org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
782 => 707 87
823 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 782
826 => org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String)
848 => org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String) 782
org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
903 => 115 27
985 => org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 707

```
998 => org.jsoup.helper.HttpConnection$Base#hasHeaderWithValue(java.lang.String,
java.lang.String) org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
1024 => 87 998 org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String)
1035 => 707 r
1069 => org.jsoup.helper.HttpConnection$Base#hasHeaderWithValue(java.lang.String,
java.lang.String) org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
1081 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 1035
1094 => 87 1069
1119 => org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String) 1035 1094
1122 => org.jsoup.helper.HttpConnection$Base#fixHeaderEncoding(java.lang.String)
org.jsoup.helper.HttpConnection$Base#looksLikeUtf8(byte[])
1170 => x org.jsoup.examples.HtmlToPlainText#main(java.lang.String[])
1176 => 11 1170
1200 => 1170 org.jsoup.Jsoup#connect(java.lang.String)
org.jsoup.helper.HttpConnection#connect(java.lang.String)
org.jsoup.helper.HttpConnection#url(java.lang.String)
1208 => 115 r
1224 => 1208 r
1237 => 1170 org.jsoup.nodes.Element#select(java.lang.String)
org.jsoup.select.Selector#select(java.lang.String, org.jsoup.nodes.Element)
org.jsoup.select.Selector#select() org.jsoup.select.Collector#collect(org.jsoup.select.Evaluator,
org.jsoup.nodes.Element) org.jsoup.select.NodeTraversor#traverse(org.jsoup.nodes.Node)
1258 => 1308 1237
1300 => 1258 org.jsoup.nodes.Node#nextSibling()
1308 => 1224 r
1323 => 1364 org.jsoup.select.NodeTraversor#traverse(org.jsoup.nodes.Node)
1334 => 115 1323
1356 => 1334 org.jsoup.nodes.Node#nextSibling()
1364 => 1170 org.jsoup.examples.HtmlToPlainText#getPlainText(org.jsoup.nodes.Element)
1374 => 1323 org.jsoup.nodes.Node#childNodeSize()
1382 => 115 1364
1393 => 1382 org.jsoup.select.NodeTraversor#traverse(org.jsoup.nodes.Node)
1413 => 1393 org.jsoup.nodes.Node#nextSibling()
1426 => 1382 org.jsoup.examples.HtmlToPlainText$FormattingVisitor#toString() 93
}
```

Résultats :

Après vérification, le résultat obtenu est conforme aux exigences de l'algorithme de Sequitur :

- La stacktrace a été compressée en passant de 1086 à 285 éléments. Soit un coefficient de compression de 73,75%.

- Chaque règle est bien utilisée au moins 2 fois

- Afin de vérifier la validité de l'algorithme, une fonctionnalité permettant de reconstituer la stacktrace de départ a été implémentée. Après avoir testé sur de multiples exemples, aucune erreur n'a été trouvée, la stacktrace après décompression a toujours été identique à celle d'origine.

Fonction main de la classe ListLinks

Le nombre d'éléments de la stacktrace de départ est de 2212 éléments, après compression la stacktrace est réduite à 426 éléments. La grammaire générée est la suivante :

S => org.jsoup.examples.ListLinks#main(java.lang.String[])
org.jsoup.helper.Validate#isTrue(boolean, java.lang.String) 11
org.jsoup.examples.ListLinks#print(java.lang.String, java.lang.Object[]) 34
org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 53
org.jsoup.Connection$Method#hasBody() 53 org.jsoup.helper.Validate#isFalse(boolean,
java.lang.String) 53
org.jsoup.helper.HttpConnection$Response#serialiseRequestUrl(org.jsoup.Connection$Request)
org.jsoup.helper.Validate#isFalse(boolean, java.lang.String) 78 97 112
org.jsoup.helper.HttpConnection#needsMultipart(org.jsoup.Connection$Request) 97 118 112
org.jsoup.helper.DataUtil#mimeBoundary() 151
org.jsoup.helper.HttpConnection$Response#initUnSecureTSL() 151
org.jsoup.helper.HttpConnection$Response#getInsecureVerifier() 151
org.jsoup.Connection$Method#hasBody() 151
org.jsoup.helper.HttpConnection$Response#getRequestCookieString(org.jsoup.Connection$Request)
229 229 247 org.jsoup.helper.DataUtil#crossStreams(java.io.InputStream, java.io.OutputStream) 284
org.jsoup.helper.HttpConnection$Response#createHeaderMap(java.net.HttpURLConnection) 284
311 339 335 311 org.jsoup.parser.TokenQueue#matchChomp(java.lang.String)
org.jsoup.parser.TokenQueue#matches(java.lang.String) 481 339 384
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 384 456 588 508 588 335
org.jsoup.helper.HttpConnection$Base#hasCookie(java.lang.String) 653 653 711
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 207
org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 711
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 790
org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 207 770 767 770 790 1038 730
org.jsoup.helper.StringUtil#resolve(java.net.URL, java.lang.String) 839
org.jsoup.helper.HttpConnection#encodeUrl(java.net.URL) 839
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request,
org.jsoup.helper.HttpConnection$Response) 839
org.jsoup.helper.HttpConnection$Response#contentType() 839 org.jsoup.parser.Parser#xmlParser()
839 org.jsoup.helper.DataUtil#getCharsetFromContentType(java.lang.String)

org.jsoup.helper.DataUtil#validateCharset(java.lang.String) 207 924 913 948 1001 948 1035 767 989
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 1001 1012
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String) 1035 1038 959 130
org.jsoup.helper.DataUtil#readToByteBuffer(java.io.InputStream, int)
org.jsoup.helper.Validate#isTrue(boolean, java.lang.String) 207
org.jsoup.helper.DataUtil#emptyByteBuffer() 118 11 1097
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 1523 1097
org.jsoup.helper.HttpConnection#encodeUrl(java.lang.String)
org.jsoup.helper.HttpConnection#encodeUrl(java.net.URL) 1121 1134 1324 1324 1390 1411 1419
1448 1448 1983 1482 1482 1641 1541 org.jsoup.nodes.Node#absUrl(java.lang.String) 1541 1792
1792 1955 2502 2363 2239 2101 2239 2363 2101 2101 2363 2502 2514
org.jsoup.nodes.Node#childNodeSize() 2524 org.jsoup.nodes.Node#childNode(int) 2544 2524
org.jsoup.nodes.Node#parentNode() 2544 1848 2570
org.jsoup.helper.Validate#notEmpty(java.lang.String) 2580 2570
org.jsoup.nodes.Attribute#getValue() 2580 2596 2622 2608 2622 2643
org.jsoup.helper.StringUtil#appendNormalisedWhitespace(java.lang.StringBuilder, java.lang.String,
boolean) org.jsoup.helper.StringUtil#isWhitespace(int) 2643
org.jsoup.nodes.TextNode#lastCharIsWhitespace(java.lang.StringBuilder) 2661
org.jsoup.nodes.Element#isBlock() org.jsoup.parser.Tag#isBlock() 2661 2666 84
org.jsoup.nodes.Element#text()
org.jsoup.nodes.TextNode#lastCharIsWhitespace(java.lang.StringBuilder) 97 x
R = {
11 => 19 84
19 => r r
34 => 11 91
53 => 78 34
78 => 19 19
84 => x org.jsoup.examples.ListLinks#main(java.lang.String[])
91 => org.jsoup.helper.HttpConnection#get() org.jsoup.helper.HttpConnection#execute()
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request)
org.jsoup.helper.HttpConnection$Response#execute(org.jsoup.Connection$Request,
org.jsoup.helper.HttpConnection$Response)
97 => 19 r
112 => 130
org.jsoup.helper.HttpConnection$Response#setOutputContentType(org.jsoup.Connection$Request)
118 => 97 r
130 => 84 91
151 => 207
org.jsoup.helper.HttpConnection$Response#createConnection(org.jsoup.Connection$Request)
207 => 296 130
229 => 247 org.jsoup.helper.HttpConnection#encodeMimeName(java.lang.String)
247 => 207 org.jsoup.helper.HttpConnection$Response#writePost(org.jsoup.Connection$Request,
java.io.OutputStream, java.lang.String)

284 => 207
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
296 => 118 97
311 => org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
org.jsoup.parser.TokenQueue#chompTo(java.lang.String)
335 => 296 97 629
339 => org.jsoup.parser.TokenQueue#consumeTo(java.lang.String)
org.jsoup.parser.TokenQueue#remainder()
384 => 407 org.jsoup.helper.HttpConnection$Base#cookie(java.lang.String, java.lang.String)
407 => 296 34
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
432 => 407 org.jsoup.helper.HttpConnection$Base#header(java.lang.String, java.lang.String)
456 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 432
481 => 335 org.jsoup.helper.HttpConnection$Response#processResponseHeaders(java.util.Map)
508 => 481 org.jsoup.helper.HttpConnection$Base#header(java.lang.String, java.lang.String)
588 => org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 432 456
org.jsoup.helper.HttpConnection$Base#removeHeader(java.lang.String)
org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 508
org.jsoup.helper.HttpConnection$Base#removeHeader(java.lang.String)
org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String)
629 => 130
org.jsoup.helper.HttpConnection$Response#setupFromConnection(java.net.HttpURLConnection,
org.jsoup.Connection$Response)
653 => 913 629 org.jsoup.helper.HttpConnection$Base#cookie(java.lang.String, java.lang.String)
663 => 296 r
711 => 767 org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
730 => 663 130
767 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 730
770 => org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String)
790 => org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String) 730
org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
839 => 118 34
913 => org.jsoup.helper.Validate#notEmpty(java.lang.String, java.lang.String) 663
924 => org.jsoup.helper.HttpConnection$Base#hasHeaderWithValue(java.lang.String,
java.lang.String) org.jsoup.helper.HttpConnection$Base#hasHeader(java.lang.String)
948 => 130 924 org.jsoup.helper.HttpConnection$Base#getHeaderCaseInsensitive(java.lang.String)
959 => 663 r
989 => org.jsoup.helper.HttpConnection$Base#hasHeaderWithValue(java.lang.String,
java.lang.String) org.jsoup.helper.HttpConnection$Base#header(java.lang.String)
1001 => org.jsoup.helper.Validate#notNull(java.lang.Object, java.lang.String) 959

1012 => 130 989
1035 => org.jsoup.helper.HttpConnection$Base#scanHeaders(java.lang.String) 959 1012
1038 => org.jsoup.helper.HttpConnection$Base#fixHeaderEncoding(java.lang.String)
org.jsoup.helper.HttpConnection$Base#looksLikeUtf8(byte[])
1097 => org.jsoup.Jsoup#connect(java.lang.String)
org.jsoup.helper.HttpConnection#connect(java.lang.String)
org.jsoup.helper.HttpConnection#url(java.lang.String)
1105 => 118 r
1121 => 1105 r
1134 => 84 org.jsoup.nodes.Element#select(java.lang.String)
org.jsoup.select.Selector#select(java.lang.String, org.jsoup.nodes.Element)
org.jsoup.select.Selector#select() org.jsoup.select.Collector#collect(org.jsoup.select.Evaluator,
org.jsoup.nodes.Element) org.jsoup.select.NodeTraversor#traverse(org.jsoup.nodes.Node)
1155 => 1419 1134
1217 => 1411 1155
1324 => 1390 1217
1390 => org.jsoup.nodes.Node#childNodeSize() 1155 org.jsoup.nodes.Node#childNode(int) 1217
org.jsoup.nodes.Node#parentNode()
1411 => 1155 org.jsoup.nodes.Node#nextSibling()
1419 => 1121 r
1448 => 84 1969
1459 => 84 org.jsoup.nodes.Node#attr(java.lang.String)
1470 => 1459 org.jsoup.nodes.Attributes#getIgnoreCase(java.lang.String)
1482 => 118 1470 org.jsoup.nodes.Attribute#getValue()
1501 => 1573 org.jsoup.nodes.Node#absUrl(java.lang.String)
1507 => org.jsoup.helper.Validate#notNull(java.lang.Object) 97
1523 => 1105 84
1528 => 1552 org.jsoup.nodes.Node#hasAttr(java.lang.String)
1541 => 1523 1528
1552 => org.jsoup.nodes.Node#attr(java.lang.String) org.jsoup.nodes.Node#absUrl(java.lang.String)
1564 => 1523 1552
1573 => 118 1459
1584 => 1507 1470 org.jsoup.helper.Validate#notEmpty(java.lang.String)
1604 => 1573 1852
1641 => 1501 org.jsoup.helper.Validate#notEmpty(java.lang.String) 1501 1899
1663 => 1564 org.jsoup.nodes.Node#hasAttr(java.lang.String)
1792 => 1819 1573 1584 1604 1604 1641 1922
1819 => org.jsoup.nodes.Attributes#hasKeyIgnoreCase(java.lang.String) 1564
org.jsoup.helper.StringUtil#resolve(java.lang.String, java.lang.String)
org.jsoup.helper.StringUtil#resolve(java.net.URL, java.lang.String) 1564
org.jsoup.nodes.Node#attr(java.lang.String)
1848 => 118 84
1852 => org.jsoup.nodes.Attributes#getIgnoreCase(java.lang.String)
org.jsoup.nodes.Attribute#getValue()

```
1863 => 1871 1852
1871 => 1848 org.jsoup.nodes.Node#attr(java.lang.String)
1882 => 1871 org.jsoup.nodes.Node#absUrl(java.lang.String)
1899 => org.jsoup.nodes.Node#hasAttr(java.lang.String) 1507 11 1528
org.jsoup.nodes.Attributes#hasKeyIgnoreCase(java.lang.String)
1922 => 1663 org.jsoup.nodes.Node#absUrl(java.lang.String) 1663
1955 => 1819 1848
1969 => 2101 org.jsoup.nodes.Element#tagName() 2666
1983 => 1459 1584
2095 => 1863 1863 1882 org.jsoup.helper.Validate#notEmpty(java.lang.String) 1882 1899 1922 1955
2101 => org.jsoup.examples.ListLinks#print(java.lang.String, java.lang.Object[]) 11
2239 => 1969 1983 2095
2363 => org.jsoup.nodes.Node#attr(java.lang.String) 1584 2095
2502 => org.jsoup.examples.ListLinks#trim(java.lang.String, int) 11
2514 => org.jsoup.nodes.Element#text()
org.jsoup.select.NodeTraversor#traverse(org.jsoup.nodes.Node)
2524 => 1848 2514
2544 => 2524 org.jsoup.nodes.Node#nextSibling()
2570 => 2583 org.jsoup.nodes.TextNode#getWholeText()
org.jsoup.nodes.Attributes#get(java.lang.String)
2580 => 1121 84
2583 => org.jsoup.nodes.Element#text()
org.jsoup.nodes.Element#appendNormalisedText(java.lang.StringBuilder, org.jsoup.nodes.TextNode)
2596 => 2583 org.jsoup.nodes.Element#preserveWhitespace(org.jsoup.nodes.Node)
2608 => 1523 2596
2622 => org.jsoup.parser.Tag#preserveWhitespace() 2608 org.jsoup.nodes.Element#parent()
2643 => 1523 2583
2661 => 1848 org.jsoup.nodes.Element#text()
2666 => org.jsoup.parser.Tag#getName() 97
}
```

Résultats :

- La stacktrace est passée de à 2212 à 426 éléments, soit une compression de 80,74%.
- Chaque règle est bien utilisée au moins 2 fois.
- La stacktrace est bien restituée après décompression.

c. Récapitulatif

Les tests de performance ont été exécutés sur un ordinateur ayant les caractéristiques suivantes :

- Java : version 8 update 121
- OS : Windows 10
- Processeur : Intel® Core™ i7-6700HQ CPU 2.60 gHZ 2.59gHZ
- Carte graphique : Nvidia GeForce 970M
- Ram : 8 go (dont 7.87go utilisable)

Ex.	Stacktrace origine	Stacktrace compressée	Nombre de règles	T (ms) Stacktrace	T (ms) Sequitur	Taux comp.
Source [1]	52	41	9	Fournie par la source	9 ms	21,15 %
Source [2]	16	10	3	Fournie par la source	3 ms	37,5 %
Source [3]	29	26	4	Fournie par la source	5 ms	10,34 %
JSoup LL	2212	426	100	4010 ms	215 ms	80,74 %
JSoup HTPT	1086	285	60	3906 ms	96 ms	73,75 %

Ex : Exemple

Taux comp : Taux de compression

VI. Critiques et améliorations

a. Critiques

PathImpact nécessite un projet contenant une seule et unique fonction main qui est le démarrage de la stacktrace. Certaines applications n'ont aucune fonction main, c'est notamment le cas des applications web.

La création de la stacktrace est beaucoup plus lente que la compression de Sequitur ; elle représente plus de 15 fois la consommation de Sequitur, voir le tableau récapitulatif de la phase d'évaluation.

b. Améliorations

Certaines améliorations peuvent être implémentées sur PathImpact. En voici quelques exemples :

- Améliorer les performances de Sequitur, par exemple, en mémorisant les digrammes déjà rencontrés dans une table de hachage.

- Étudier la possibilité de diminuer la taille de la stacktrace de départ tout en conservant les mêmes informations.

- Implémenter la création du DAG via la grammaire pour déterminer les impacts.

- Lorsqu'une méthode appelle une méthode abstraite, il faudrait faire une analyse complexe consistant à retrouver l'origine de cette variable et se poser les questions suivantes « Où la variable est-elle valorisée ? Quel est le type réel de l'objet ? ». Cela serait cependant difficile à mettre en place, car il faudrait parcourir toutes les couches du programme et gérer énormément de cas (il est même possible que des données fournies en paramètres ou d'une base de données influent sur le type…)

VII. Conclusion

Au fur et à mesure que les applications évoluent, certaines fonctionnalités peuvent être amenées à être modifiées ou supprimées, que ce soit pour une non-utilisation, simplification, évolution etc.

Cependant, des impacts dont leur ampleur est bien souvent difficile à déterminer peuvent survenir. Effectuer un chiffrage avant d'effectuer les développements pour estimer le coût, ou tout simplement s'y retrouver dans les multiples classes impactées lors des développements peut alors devenir un véritable problème.

De multiples programmes permettent d'analyser les impacts si une méthode donnée est modifiée, et PathImpact en fait partie.

PathImpact s'exécute en trois parties, plus précisément :

- Une génération de stacktraces concaténées.
- Une compression des stacktraces via Sequitur (compresse en créant des règles de grammaire qui permettent d'éliminer les doublons de digramme)
- Une construction d'une structure d'arbre générée par la grammaire compressée qui permet de déterminer les impacts.

Cet algorithme est très précis dans l'analyse d'impacts, mais est l'un des analyseurs les plus consommateurs de ressources, en termes de mémoire, calculs et temps d'exécution.

Les deux premières parties de PathImpact ont été implémentées, testées et validées sur des petits exemples, ainsi que sur un projet de taille réel (JSoup).

Les prochaines étapes consisteront à implémenter la dernière partie de PathImpact, optimiser les traitements, et analyser la cohérence des impacts sur un projet de taille réelle.

Par la suite, il serait intéressant d'implémenter l'algorithme CoverageImpact [1], un autre analyseur d'impacts beaucoup moins coûteux en termes de temps d'exécution mais moins précis (n'utilise pas la compression), et de comparer les résultats obtenus. Un analyseur hybride pourrait ensuite, comme le propose la source [1], voir le jour ce qui permettrait de conserver la même précision que PathImpact tout en ayant des réponses plus rapidement.

VIII. Sources

[1] An Empirical Comparison of Dynamic Impact Analysis Algorithms
http://www.cc.gatech.edu/~orso/papers/orso.term.harrold.ICSE04.pdf

[2] CSEP 590 Data Compression, Autumn 2007 – University of Washington
https://courses.cs.washington.edu/courses/csep590a/07au/lectures/lecture05small.pdf

[3] Incremental Dynamic Impact Analysis for Evolving Software Systems
https://cse.unl.edu/~grother/papers/issre03.pdf

[4] Sequitur – University of Waikako
http://www.sequitur.info/